浙江大学

程 序 设 计 专 题

大 程 序 报 告

大程名称：基于语义分析的简单浏览器

2021~2022 春夏学期

目 录

目录

简易网页浏览器大程序设计项目

1 大程序简介

1.1 选题背景及意义

网页浏览器在日常生活中应用得相当多，每天我们在访问网页时，打开一个搜索引擎都会看到相似的界面、相似的功能。例如：通过网址的输入栏打开需要的网页；标签栏显示已经打开的网页；前进后退按钮实现网页路由；以及搜索历史、菜单等……。这些我们每天都在用的东西是否能由我们自己用已有的知识来实现呢？看起来复杂的浏览器是否能由我们自己亲手做出？秉承贴近生活和应用范围广阔的理念，我们组选择了制作简易浏览器这一题目。

本次选题基于语义分析，工程分工较为方便，适合由小组成员合作完成，各自找到适合自己、匹配水平的任务；题目涉及知识应用性广，最终成果直观，非常适合锻炼个人能力，激发对计算机科学的热情。

1.2 目标要求

- 实现网络路由，包括网页的前进、后退、主页、刷新、输入地址并跳转；
- 支持两个模式：全屏显示与分屏（分别显示源代码与渲染结果）；
- 有限的样式/CSS 渲染支持；
- UI 美观，易于操作；

- 能够识别转义符号；
- 网页内与跨页锚点定位跳转；
- 运用要求的数据结构；
- 实现其他额外高级功能等。

1.3 术语说明

- LL(1)文法：从左到由扫描输入串，采用最左推导，最多前瞰一个字符就可以决定状态转移。用来对 HTML 进行词法分析。
- Lexer：词法分析器，将 HTML 的元素拆解为上下文无关的元素。
- Token：也即 Lexer 得到的元素，类比英语中的单词，具有词性、拼写。
- Token Stream：整个 HTML 文档得到的 token 构成一整个 token stream.
- Parser：语法分析器，将 Token 进行上下文相关的分析并进行状态转移，让程序“读懂”Token Stream.类比对已经区分出单词的英语句子进行语法分析。
- DOM：Document Object Model，具体形式是 HTML 解析生成的一颗抽象语法树（AST）.每个节点有不同的类型与内容，组合成整个 HTML 文档的层次性架构，类比英语句子主谓宾、从句的层次结构。内容储存在最底层的“叶子节点”处。
- DFS：深度优先遍历，对 DOM 树进行 DFS，DFS 序即为原文文字出现顺序。
- MVC：一种设计模式/架构，由 Model、View、Controller 组成。Model 负责数据运算,View 提供用户 UI 视图,Controller 控制程序运行。三者低耦合。
- 内联函数：用 inline 修饰符修饰的函数，就地展开，不参与链接；类似宏，但更少多义性，有类型检查，该特性从 C99 标准开始被支持。
- 哈希表：散列表，通过哈希函数 pseudo-唯一地决定字符串映射下标。
- 单元测试(unit testing)：对软件中的最小可测试单元进行检查和验证。
- 耦合：指两个或两个以上的体系或两种运动形式之间通过各种交互作用而彼此影响，此处一般指的是程序各组件直接相互依赖的程度。低耦合有利于调试与单元测试。本次程序编写后期的一大工作就是进行解耦。

2 需求分析

2.1 功能需求

一、分屏模式与全屏模式的切换：

①分屏模式：该模式下分为左右两个窗口，左侧为源文本，右侧为 HTML 解析得到的渲染结果；

②全屏模式：仅保留渲染结果，类似日常使用的网页浏览器。

二、CSS 样式：支持读取 style 标签内容，以及标签 style 属性定义的 CSS 样式；

支持简单类选择器/ID 选择器；
三、多标签页：能够同时并存地访问多个窗口；
四、路由功能：前进后退以及刷新，即时反映本地文件变化；

2.2 数据需求

2.2.1 IO 需求

- 输入数据
  HTML 的合法文档，为高级错误处理通过在 token 中预留了行号、列号，为更精确的错误处理做好准备。

[约束]
该文档的语法必须合法；为了实现对中文的支持，**需要以 ANSI 编码格式提供文件（记事本打开→Ctrl+Shift+S 另存为，选择 ANSI 编码覆盖保存）。**
HTML 在文件中的形式要经过数次转换：文本字符串→(Lexer)→Token 流→(Parser)→DOM（AST）→(与 CSS 结合)→渲染树(Render Tree)。
通过编辑 initial.css 可以控制默认 CSS 设置。

- 输出数据
  通过在渲染树上进行 DFS，期间不断把节点压入栈中直到宽度超出渲染区域宽度，此时栈中元素即为该行输出的元素。遍历栈中元素，获得最大高度，计算字体对齐基准线（Baseline），作为一行输出。
  渲染结果经过主渲染器呈现在渲染区域中。

2.2.2 数据结构

- DOM：抽象语法树 AST，多叉树嵌套结构体；
- RenderTree：多叉树，嵌套结构体；
- 标签页：**双向链表**，快速插入删除；
- DOM 元素属性(attribute)：哈希表，实现快速字符串-字符串映射
- 内存池：带对齐的数组，迅速寻址与底层访问加速，提高缓存命中率，统一化内存管理，为将来更先进的内存与 GC 管理留好接口；
- 使用二进制读方式读入 html 文件，并通过文件指针获取文件内容；

2.2.3 自定义类型

DOM 与 Render Tree 两者内禀地具有递归结构，**遍历与修改都通过递归完成**；同时，这些结构的定义大量使用了**高级指针**，通过类型系统上的抽象使得程序更具有可读性(readability)；

2.3 性能需求

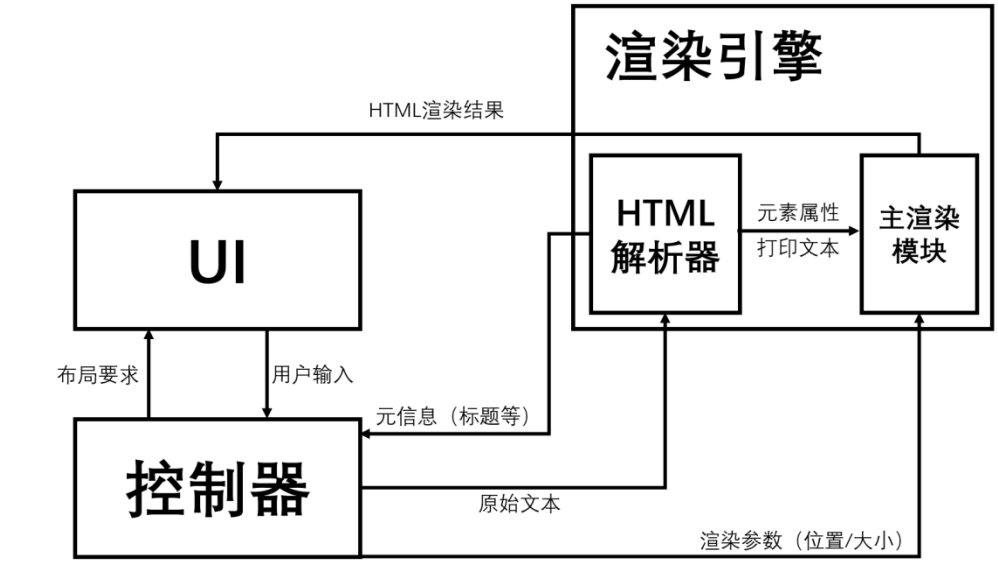
- 用户操作流畅，没有明显卡顿；滚动顺滑；

- 没有明显的内存泄漏问题，鲁棒性高；
- 减少无谓的文件读取，尽可能在程序中缓存以加快效率；
- 能够应对大文件的读入与渲染；
- 通过内存管理高效应对频繁的小块内存需求。

3 程序开发设计

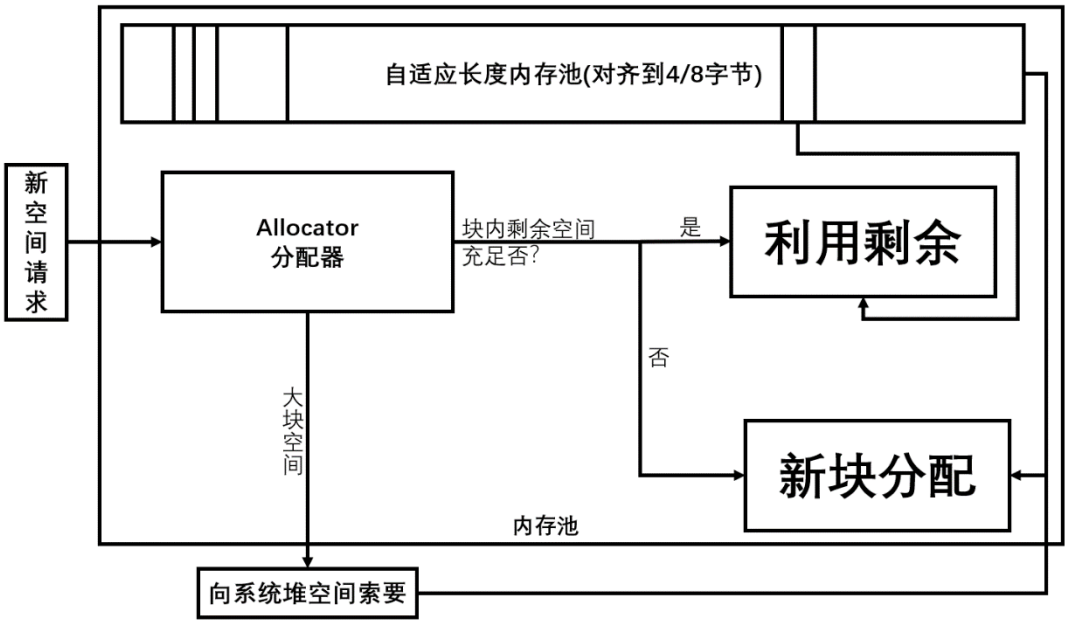
3.1 总体架构设计

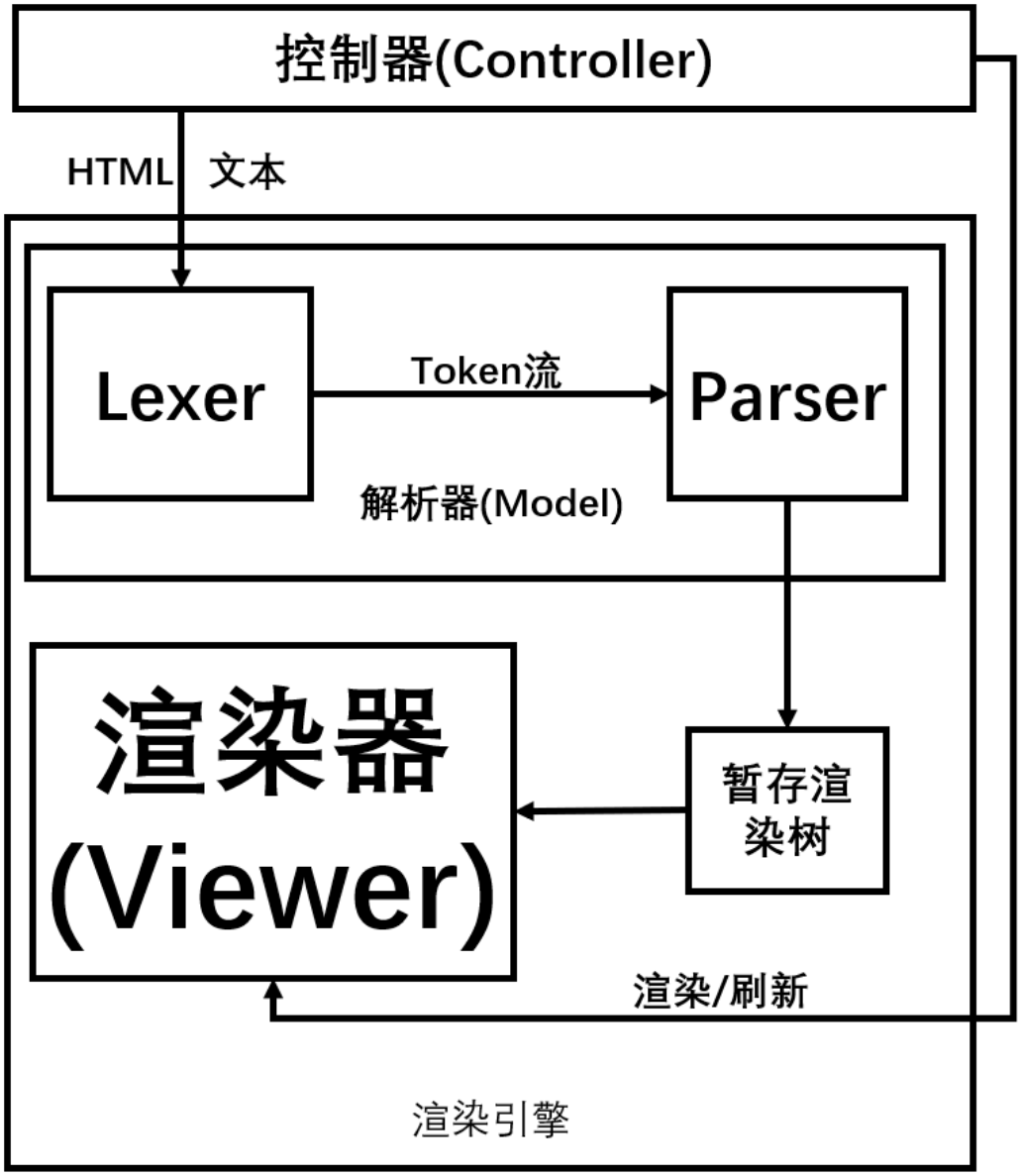
程序总体架构采用**类 MVC 的设计模式**。其中 UI 作为 Viewer 负责前端展示，渲染引擎作为 Model(Data Handler)解析 HTML 并建立抽象语法树；这两者由控制器事件驱动地进行调配。

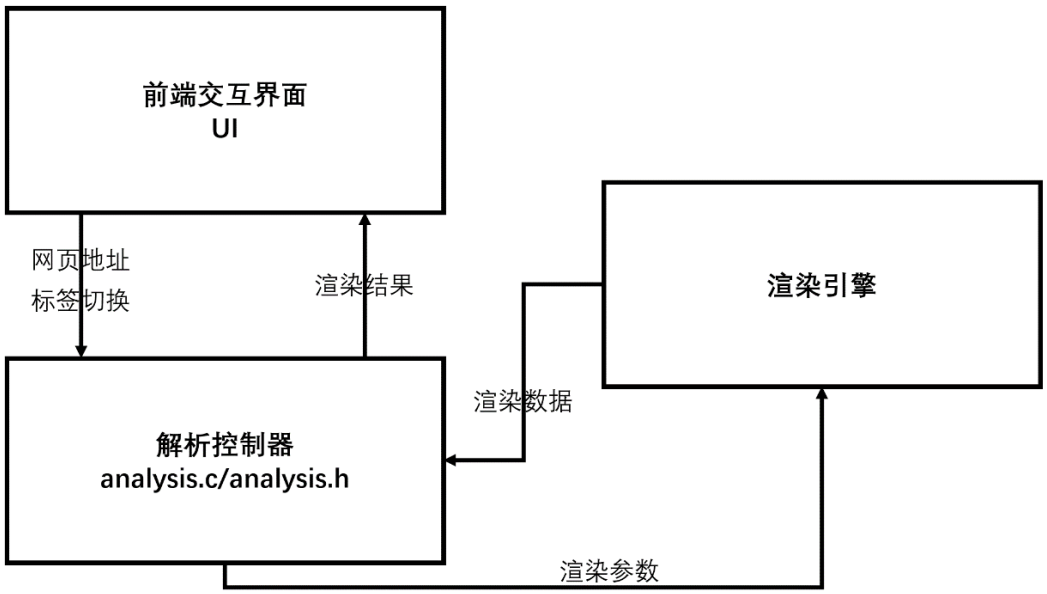


3.2 功能模块设计

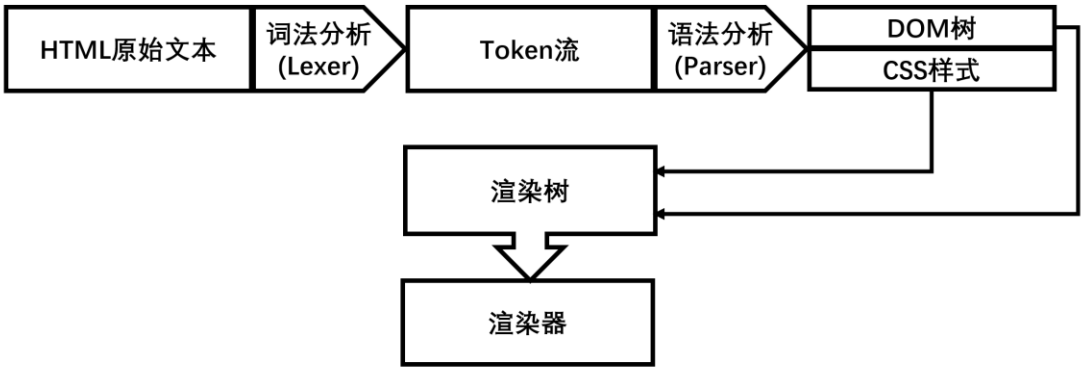
UI 分为背景/按钮等部分，由下至上逐层绘制；
后端主要由渲染引擎调用。


自适应长度内存池(对齐到4/8字节)
新空间请求
Allocator
分配器
块内剩余空间
充足否?
是
利用剩余
否
大块空间
新块分配
内存池
向系统堆空间索要


控制器(Controller)
HTML
文本
Lexer
Token流
Parser
解析器(Model)
渲染器
(Viewer)
暂存渲染树
渲染/刷新
渲染引擎



后端数据流动遵循了编译器的实现方式：



3.3 数据结构与变量设计

重要数据结构与配套算法

- DOM：抽象语法树 AST，多叉树嵌套结构体(大量嵌套/高级指针)
- Render Tree：多叉树，嵌套结构体
  以上两者内禀地具有递归结构，遍历与修改都通过递归完成。
- 标签页：双向链表，快速插入删除
- DOM 元素属性(attribute)：哈希表，实现快速字符串-字符串映射
- 内存池：带对齐的数组，迅速寻址与底层访问加速，提高缓存命中，统一化内存管理。

重要宏

```
#define SAVEPEN(x)  int x##style__=GetStyle(),x##point__=GetPointSize()\
,x##pen__=GetPenSize();string x##color__=GetPenColor();

#define RECOVERPEN(x)  SetStyle(x##style__);SetPointSize(x##point__);\
SetPenColor(x##color__);SetPenSize(x##pen__);

#define RN(x) ((x)==NULL?"":(x))
#define getAttr(x,y) ((x)->named_attributes[hash(y)])
#define GetFontWidth() TextStringWidth("_")
```

- SAVEPEN(x)：保存字体大小颜色信息，标识符 x
- RECOVERPEN(x)：恢复标识符为 x 的字体大小颜色信息
- RN(x)：将空字符串指针替换为空字符串
- getAttr(x,y)：提取 x 标签的 y 属性
- GetFontWidth()：获取单字符宽度（如果为等宽字体）

全局变量含义

`int TabNumber;`
当前第几个 Tab

`int TotalTabNumber;`
此时共几个 Tab

`int pageflash;`
页面是否需要刷新标志

`int newflag;`
是否简历链表新节点的判断

`char textbuf2[MAX][MAX];`
全局变量存放网址（文件名）

`char add[MAX];`
textbox 输入内容缓冲区

`string title;`
提供标题拾取支持。

`string globalCSS;`
全局 CSS 字符串

`string redirect;`
提供重定向选项支持

`string html_str;`
渲染器的 HTML 文件字符串原始输入

`double WinWidth,WinHeight;`
窗口长宽，用于元素的**相对定位**；

float_64 shift_x,shift_y;
HTML 渲染时的坐标系偏移量，用于滚动;

bool lazying;
标明 HTML 是否处于“惰性”状态，利用已经计算好的 DOM 树 Render Tree.

P_ERROR err;
指示解析器(Lexer&Parser)的错误状态代码。

int TabNumber;
标签页的序号，表示当前是第几个标签页。

int TotalTabNumber;
当前标签页的总数目。

int pageflash;
判断是否解析文件，0 代表未解析，1 代表解析。

int newflag;
判断是否建立链表新节点，0 代表未建立，1 代表已建立。

int color;
代表页面色系，1 代表蓝色系“sea”，0 代表黄色系“sun”。

int style;
表示页面风格，1 代表分屏模式，0 代表全屏模式。

char textbuf2[MAX][MAX];
用于存放网址全局变量。

char add[MAX];
临时储存网址的数组。

FILE *fp;
文件指针，指向要打开的网页

3.4 部分重要函数设计描述（修改库函数已标红）

static inline uint_32 hash(string s);
简单字符串哈希函数，用于在散列表中储存属性字符串指针。
位于 lexerParsr.h

TOKEN* htmlLexer(string s);
提供 html 字符串，转换为 token 流返回。同时**处理转义字符等。**
位于 lexerParser.c

TOKEN * tagParser(TOKEN *s, DOM *father);
对 token 流进行分析，处理标签层级结构，构建 DOM 树返回。
位于 lexerParser.c

DOM* htmlParser(string s);
将 html 字符串转为 DOM 树。
位于 lexerParser.c

RENDER* buildRenderTree(DOM *root,PRINT_OPT opt);
给定 DOM 树与初始打印选项构建渲染树。
树中包含了各个标签节点对应的样式与相对位置信息。
位于 renderEngine.c

---

void mainRender(float_64 x,float_64 y,float_64 w,float_64 h,RENDER *root);
主渲染器，给定渲染区域位置大小信息与渲染树，进行 dfs 遍历渲染；
位于 renderEngine.c

RENDER *findRenderByID(string id, RENDER *node);
通过 ID 定位元素，为页面跳转提供支持。输入字符串与 id 以及 DOM 树根，返回找到的节点指针（NULL 代表未找到）。
位于 renderEngine.c

---

void jumpByID(string id);
给定元素 ID，跳转到元素位置。
实现了对锚点定位符的支持。
位于 renderEngine.c

---

void applyPrintOpt(PRINT_OPT opt);
给定 PRINT_OPT 类型的打印/渲染选项，使其切换到这个变量标示的画笔状态（例如加粗下划线等）。
位于 renderEngine.c

---

static inline uint_32 str2color(string s);
[内联函数]提供 16 进制颜色编码的转换，位于 renderEngine.h.
输入 16 进制格式字符串（带#或不带均可），输出整数颜色值。

---

void addinput();
输入框函数，输入相关网址
位于 tabs.h

---

void Color();
自定义颜色的集合函数，定义了两套颜色，每套各 6 中颜色。
主要用于 UI，调节页面颜色。
保存在 color.c 和 color.h 文件中。

---

void colorStyle();
两套颜色之间的自由切换函数，内部设置一个按钮，可以在“sea”色系和“sun”色系之间切换。
主要用于 UI，切换页面颜色。
保存在 color.c 和 color.h 文件中。

---

void drawStyleButtons();
分屏和合屏函数，分屏模式下分为左右两个窗口，左侧为源文本，右侧为 HTML 解析得到的渲染结果；全屏模式下，仅保留渲染结果，类似日常使用的网页浏览器。主要用于 UI，切换页面风格。

---

void drawTabs（）；
画标签页并实现“+”和“x”
通过局部变量 i 和 k 循环画 Tab，通过 TAB_TOTAL 记录总共标签页个数，测试前后链表每个节点的复制情况，点击“x”按钮，实现后面链表内容复制到前面

---

void addNodeToLink()；
新建链表节点函数保护，输入对的网址成功打开文件并新建节点，若输入错误或者输入为空，不建立新节点

---

void analysis()；
解析文件函数，调取渲染引擎进行渲染，同时解析普通文件，用于后面标签页和网页路由的实现和调试，

---

void display（）；
刷新页面
位于 main.c

---

void Button()；
前进后退按钮的实现，通过双向链表和鼠标响应函数来联系 analysis 函数去解析文件，并优化了按钮图标
位于 button.h

库文件修改报告

`void partialDisplayClear();`

仿照库里函数编写（位于 graphics.c 中），只刷新需要刷新的部分而不是全页面刷新

位于 graphics.c

`int button(int id, double x, double y, double w, double h, char *label);`

修改的 imgui.c 函数 button，进行 button 响应样式的改变，由弹出移动位置矩形变为填充矩形闪烁，，按钮样式改变，更加美观

位于 button.h

`int textbox(int id, double x, double y, double w, double h, char textbuf[], int buflen);`

修改的 imgui.c 函数 textbox，新增了对回车键的响应，对接 analysis.c

```
typedef struct {
    void *(*allocMethod)(size_t nbytes);
    void (*freeMethod)(void *ptr);
    void (*protectMethod)(void *ptr, size_t nbytes);
} _GCControlBlock_self;
typedef _GCControlBlock_self *_GCControlBlock;
```

修改了 gcalloc.h 中内存池的定义，便于对接

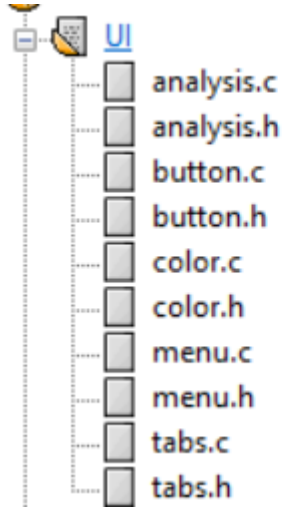
```
extern _GCControlBlock arenaInit();
#define ARENA_ENABLED 1
   if (ARENA_ENABLED && _acb==NULL)
       _acb=(_GCControlBlock)arenaInit();
```

修改了 genlib.c 中的 GetBlock 函数，完成与自己实现的内存池对接

以及其他一些为了正确包含与链接而添加的头文件与 extern 声明。

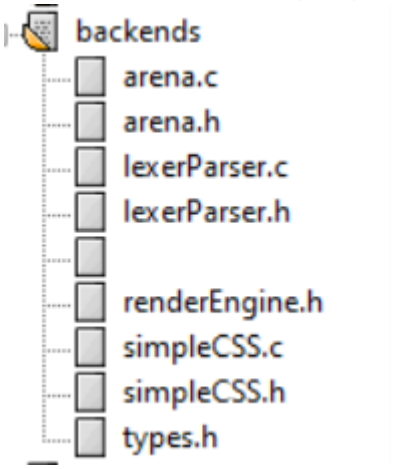
3.5 源代码文件组织设计

1）文件函数结构
文件夹 UI：



<analysis.h><analysis.c>主要渲染器
<button.h><button.c>按钮功能组件支持
<color.c>颜色功能模块，提供多色系支持
<color.h>颜色支持基础结构类型声明
<tabs.c>标签页（基于双向链表的实现）

文件夹 backends（后端支持）：



<arena.c>内存池具体操作实现
<lexerParser.c>HTML 语法解析模块
<lexerParser.h>>HTML 语法解析声明，DOM 树基础类型、结构定义，错误支持
<renderEngine.c>渲染主引擎，提供主要渲染函数与字体样式支撑组件：

```
void mainRender(float 64 x,float 64 y,float 64 w,float 64 h,RENDER *root);
void drawHtml(float 64 x,float 64 y,float 64 w,float 64 h);
```

<renderEngine.h>渲染树基础类型、结构定义
<simpleCSS.c><simpleCSS.h>简单 CSS 解析、CSS 选择器支持组件
<types.h>全局抽象/高级/嵌套类型声明，例如 bool 与 string

2）多文件构成机制

（部分重要/复杂头文件）包含关系：

<main.c>

```
#include <strlib.h>
#include <stdio.h>
#include "graphics.h"
#include "extgraph.h"
#include "imgui.h"
#include "color.h"
#include "button.h"
#include "tabs.h"
#include "renderEngine.h"
#include "types.h"
```

<tabs.c>

```
#include <strlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>

#include "graphics.h"
#include "extgraph.h"
#include "imgui.h"

#include "color.h"
#include "tabs.h"
#include "renderEngine.h"
```

<types.h>

```
#include <stddef.h>
#include "boolean.h"
```

<button.c>

```
#include <strlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>

#include "graphics.h"
#include "extgraph.h"
#include "imgui.h"

#include "color.h"
#include "button.h"
#include "tabs.h"
#include "renderEngine.h"
```

所有头文件均采用宏定义保护头，防止重复包含。

通过分析头文件依赖链，使重复调用覆盖范围最少。

4 部署运行和使用说明

4.1 编译安装

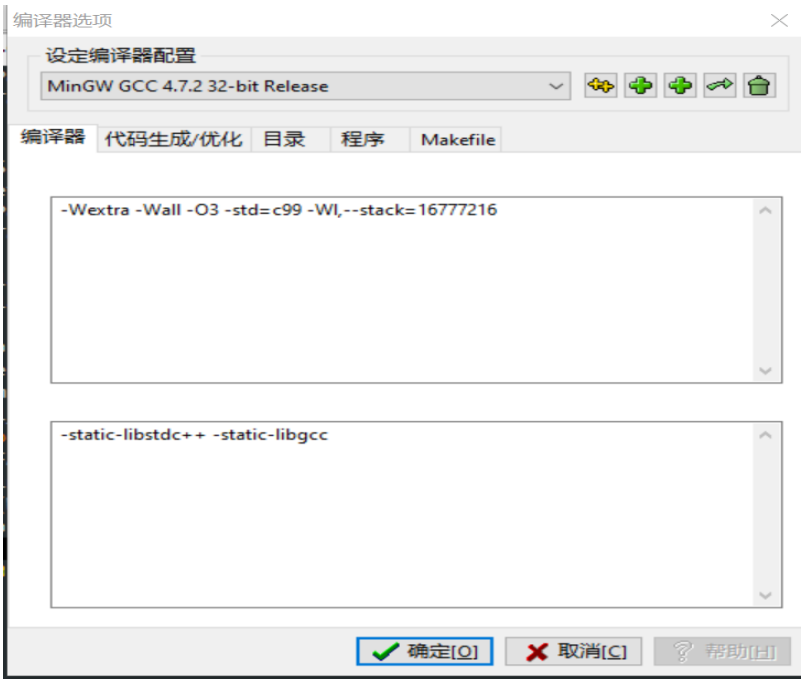
在 Dev C++环境下编译安装，编译器为MinGW GCC 4.7.2 32-bit Release。

先尝试直接编译运行，若无法通过，才需执行以下步骤：

【极为重要】由于使用了inline关键字（C99标准加入，ANSI-C/C89 标准无法识别），获取源代码包后请务必在工具>编译选项>勾选编译时加入以下命令，填入：

```
-O3 -std=c99 -Wl,--stack=16777216
```

如下图所示(如遇到问题，可在钉群咨询)：



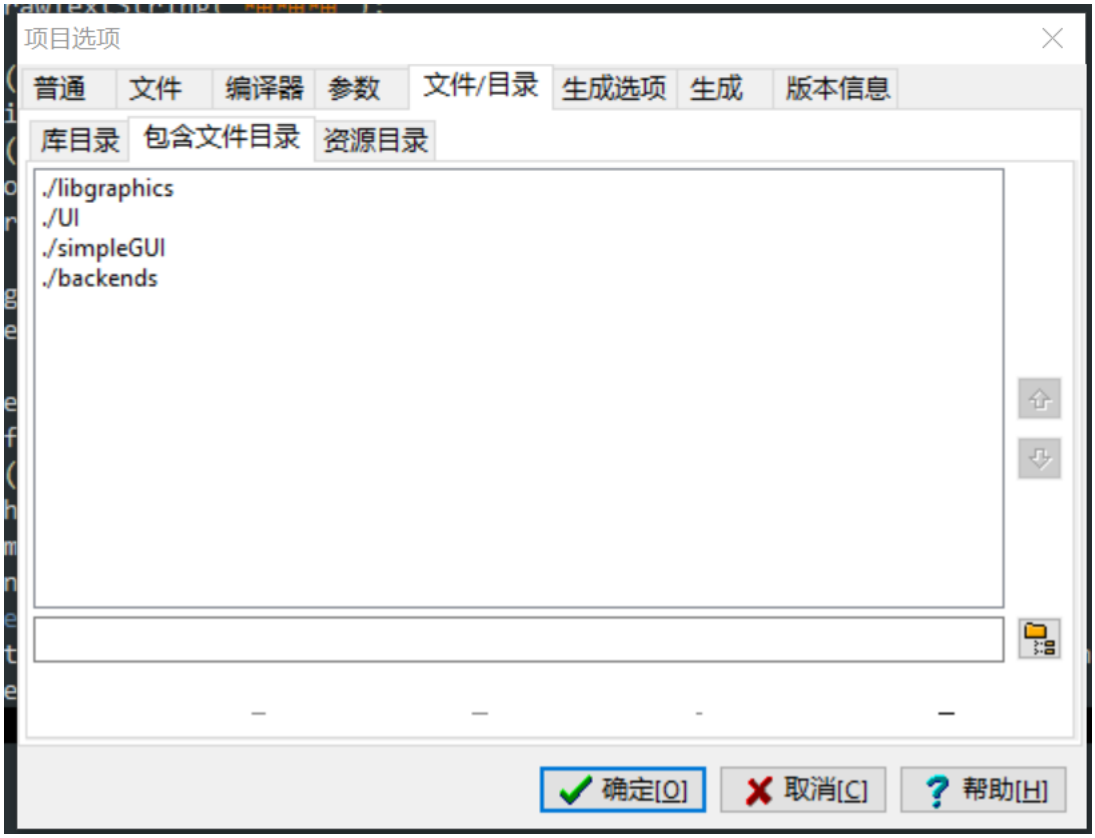
同时在项目>项目属性>参数>附加的命令行选项中选择 C compiler，填写：

```
-O3 -std=c99 -Wl,--stack=16777216
```

并保存。

在项目>项目属性>文件/目录>包含文件目录中添加各个子文件夹(如下图所示)：

```
./libgraphics
./UI
./simpleGUI
./backends
```



如果编译出现错误，返回这一步，将相对路径改为手动选择文件夹的绝对路径逐个添加。

此后同样在工具>编译选项>目录>C 包含文件中添加文件夹，分别为源代码包目录下的：libgraphics 文件夹、simpleGUI 文件夹、backends 文件夹、UI 文件夹

4.2 运行测试

4.2.1 主要调试手段

- 利用 Dev C++自带的 GDB Debugger，通过下断点、单步跟踪对局部进行跟踪监测矫正错误；
- 对于函数进行单元测试、等效替代、对拍等，保证函数副作用最小化，不对运行环境变量产生非法影响，降低耦合；
- 谨慎使用全局变量传递参数，降低耦合，提高封装程度；
- 严格控制内存非法访问，妥善处理 NULL 指针；
- 通过 InitConsole()打开命令行窗口调试，与 gdb 等 Debugger 有机结合，灵活运用，知晓程序状态，判断程序是否运行符合预期；

- 必要时，通过汇编代码找到越界点，与已有变量地址比较从而定位关键变量、关键函数位置，找到错误点、泄漏点进行矫正；
- 在编译选项中增加-Wall 与-Wextra，丰富错误信息，规范编码；
- 通过 memleak 工具判断内存泄漏点，解决自身失误导致的内存泄漏；
- 通过 perf 生成火焰图判断性能瓶颈。

4.2.2 实际测试

1.在实“标签页的关闭”中，考虑到不同标签页（不同双向链表）里面的页面个数（链表节点）不同，还可能有空页面（blank），需要测试前后链表每个节点的复制情况，点击“x”按钮，实现后面链表内容复制到前面，测试点：

1） 前链表长度比后链表短：前链表为空；前链表非空；

2） 前链表长度比后链表长：后链表为空；后链表非空；

3） 两条链表长度相等。

（注:在实现链表复制的过程中,需要及时 free 掉多余节点,及时建立新的节点,我们可以通过新建链表节点函数输入网址名作为形参,判断是要 free 还是新建）

2.在解析文件内容时，如果输入网址正确，则成功打开文件并新建链表节点；若输入网址错误，需要提示操作者输入错误，不建立新节点；输入为空（直接按回车），需要解析成新建窗口，不建立新节点通过 PAGE_FLASH 和 NEW_FLAG 进行判断。

3. 分装函数时经常出现多重定义和找不到 GUIID，头文件保护并不能很好的解决多重定义问题，需要再进行细致优化，而 GUIID 我们会重新复制放入目标文件.h

4. 网页的刷新，imgui 库里的 textbox 函数并不能对回车进行响应,经修改后，我们通过全局变量存储输入的网址，并判断其为正确网址的情况下放入相应标签页的链表结点

5. 网页的刷新，我们添加 partialDisplayClare 函数实现仅仅刷新我们想要刷新的部分，我们通过全屏画点图，调试参数看页面刷新相应变化来准确地找到不同参数对应的区域

6. 标签页的刷新，需要在无标签页时留有“+”号，在填满六个标签页时隐藏“+”

   测试点:

   1） 无标签页，点击“+”

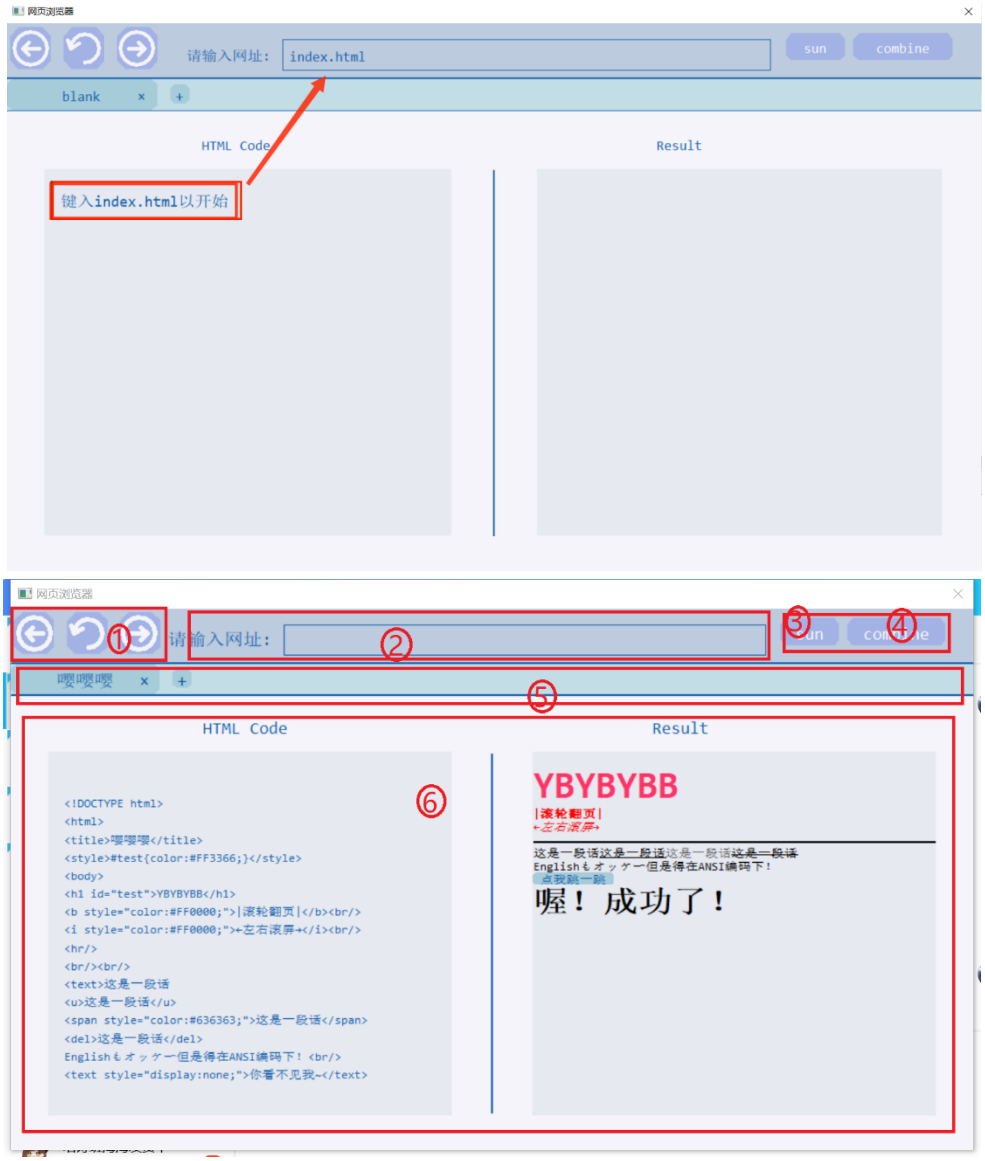
   2） 有标签页，小于六个，点击“+”和“x”

   3） 有标签页，填满六个，点击“x”

为了更好更全面地测试程序，预制了一些网页用于显示，可以在根目录下找到。

4.3 用户使用手册

打开程序，根据提示在地址栏键入 index.html [回车]，开始读取。



① 右上角的三个图标从左至右分别是前进、刷新、后退按钮，用户在同一个标签页内可以通过前进和后退访问在当前网页之前打开的网页和之后打开的网页，通过刷新按钮可以重新加载页面。

② 网址的输入框，在这个文本框里输入要访问的网址，再按回车键，就可以在 6 区域看到网页的内容。

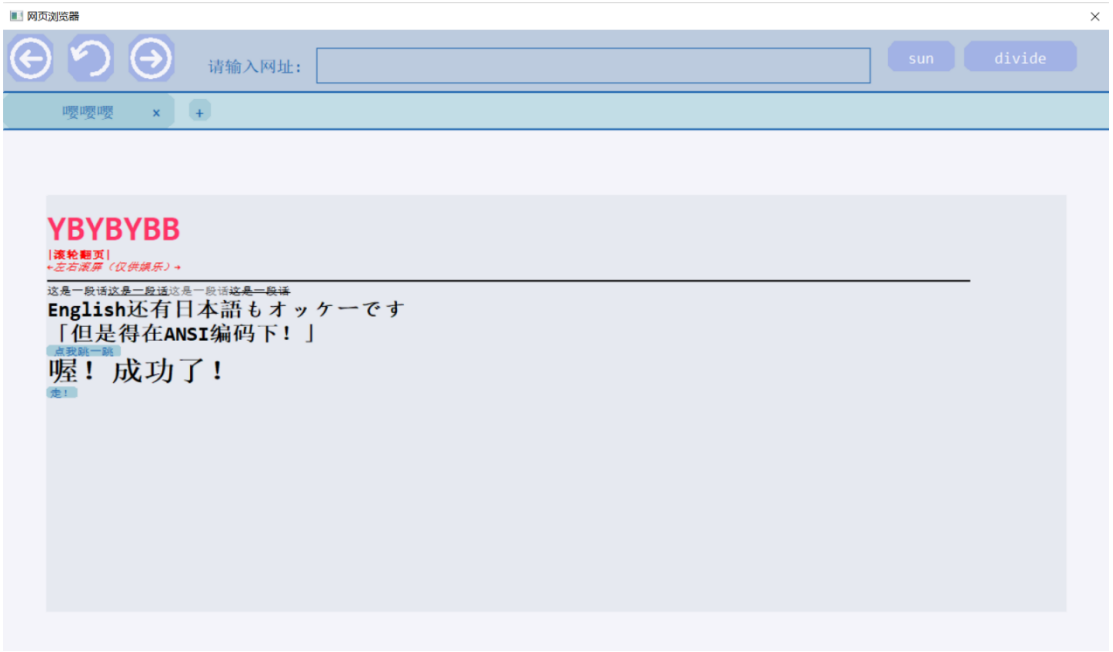
键入 index.html 后的页面：

③　页面的颜色切换按钮，浏览器的颜色搭配分为两个色系：蓝色系和黄色系，分别对应自然界中的“sea”和“sun”，蓝色使人冷静、理性，黄色让人热情、开心，这可以调节用户使用浏览器时的情绪，满足用户不同的工作任务需求。

（sun 色系截图）

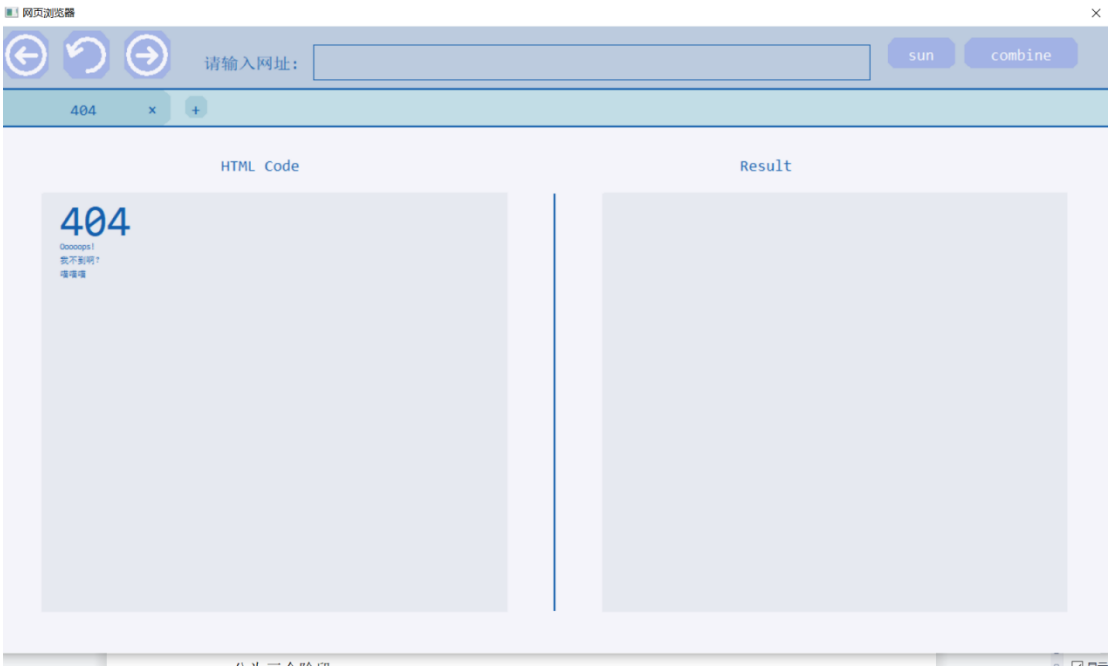
④　页面的分屏与全屏按钮，可以调节浏览器显示页面的样式是分开的左边显示源码，右边显示渲染结果，还是合并的整个屏幕上显示渲染结果。

（下图为全屏时的效果）



⑤ 标签栏，总共能分布 6 个标签页，标签栏中央显示着打开的网页名字，如果网页不存在，则显示 404，表示错误。每个标签页右端有“×”记号，点击表示关闭该标签页，右侧有“＋”记号，表示在该标签页的右边再增加一个新的标签页。当增加到 6 个标签页时，“＋”号自动隐去，无法再添加新的标签页。初始时，标签栏中会有一个标签页，用户可以删去这唯一一个标签页，此时标签栏中没有标签页，只留下“＋”号；也可以不断增加标签页，直到满六个为止。

输入错误的网址后：



⑥ 主要显示区域：网页的解析结果将在这里显示，通过分屏和全屏按钮，可以变换显示的模式。

5 团队合作

5.1 开发计划

分为三个阶段：
1 前期：确定分工，各成员进行背景知识搜集学习，确立编码规范；
2 中期：着手编写分别完成各模块代码，进行单元测试；
3 后期：打包封装，整合微调，部分代码拆分或重构，撰写报告，录制视频。

5.2 建议编码规范

- 普通变量名全小写，采用下划线分割：var_name_example
- 常量名全大写，采用下划线分割：CONSTANT_NAME_EXAMPLE
- 函数名采用小驼峰命名法：functionNameExample
- **每个.h 头文件需要能够独立编译成功；**
- 头文件的包含遵循最小原则：在能够编译的情况下**包含的头文件范围最少**（这需要小心分割程序，并**绘制头文件的依赖链，选择合适的进行包含。**例如对于依赖关系 C→B→A 与 D→B，A 中只需要包括 B 即可）；
- **头文件必须具有保护头；**
- **宏定义与内联函数（inline）应该放置于.h 头文件，而非.c 头文件中；**
- **推荐优先包含标准库文件；**
- **花括号换行于：函数**
- **花括号不换行于：if/while/for/struct/enum**

5.3 代码任务分工

本次大作业总代码行数：1664 行。由于存在共同完成的函数，各组员代码行数相加可能超过总行数。

5.3.1 A

负责 HTML 在 LL(1)文法下，通过词法分析器 Lexer 先转为 token stream，再经过语法分析器 Parser 到 DOM 树的解析转换流程；
将 DOM 树结合简易的 CSS 识别生成渲染树 Render Tree，决定元素在屏幕上的位置从而绘制图案；
实现了 CSS 从父级的继承以及全局类选择器、ID 选择器；
实现了通过内嵌 css 指定 16 进制颜色、下划线、斜线、粗体、超链接、页面锚符号，各级标签与一些额外功能；

实现了页面的滚动（含左右，仅供演示）；
额外实现了 Arena 内存池，并通过 libgraphics 预留接口接入。
参与撰写润色实验报告，辅导组员调试并解决问题，统筹成员进度，布置任务；
具体编写：文件夹 backends 中的所有文件，与 analysis.c 中的 analysis 函数。
共完成代码约 1030 行。

5.3.2 B(UI 界面设计实现)

设计页面布局和颜色，实现分屏和合屏的切换；
实现了多配色方案的切换；
撰写实验报告。
具体分工：main.c 文件、analysis.c 文件、color.c 文件、button.c 文件 1-43 行、button.h 文件、color.h 文件、tabs.h 文件，**共完成代码约 483 行**。

5.3.3 C(网页路由与页面组件)

与另外两位共同完成 UI；
具体分工：main.c 文件、tabs.c 文件、analysis.c 文件、button.c 文件 48—111 行、analysis.h 文件，**共完成代码约 525 行**。

5.4 报告分工

A

- 章节一大程序简介中的 1.2 目标要求，1.3 术语说明；
- 章节二需求分析的全部内容；
- 章节三程序开发设计中的 3.1 总体架构设计，3.2 功能模块设计，3.3 数据结构与变量设计中的高级数据结构与宏；
- 章节四部署运行和使用说明中的 4.1 编译安装与 4.2.1 主要调试手段；
- 章节五分工合作中的 5.1 开发计划，5.2 建议编码规范，5.3 代码任务分工，5.4 其他分工， 5.6.1 开发亮点的全部内容以及 5.6.2 开发历程中的文字部分；
- 章节七参考资料的全部内容；
- 对报告进行润色。

B

- 章节一大程序简介中的 1.1 选题背景及意义；
- 章节四部署运行和使用说明中 4.3 用户使用手册的全部内容，合作完成 4.2.2 实际运行测试；
- 对报告进行润色、排版修正、提供大量富有建设性的修改建议；

C

- 合作完成 4.2.2 实际运行测试；
- 为 5.6.2 开发历程提供截图；

此外，各组员都：

- **共同参与完成 5.3 代码任务分工，3.4 数据结构与变量设计，3.5 源代码文件组织设计中对应自己撰写代码部分的内容；**
- **共同参与完成 6 收获感言中对应自身的内容。**

5.5 其他分工

A

- PPT 制作；
- 验收上台介绍；
- 验收后端演示；
- 视频录制与调整；
- 实验报告的润色；

B

- 程序验收演示；
- 后期代码拆分与再封装；

C

- 暂无。

5.6 个人遇到的难点与解决方案

A

1）strlib 提供的函数虽然方便易用，但在堆空间中不断申请细小内存空间，产生了大量零碎字符串。这些“内存碎片”大大削弱了程序的空间局部性，造成缓存命中率大幅降低。

解决方法：其实给定的库中有尚未实现的内存池接口。我对比了不同的内存池实现方式（包括 SGI STL 的空间配置器），最终选择简化实现了谷歌主导的 LevelDB **中使用的 Arena 内存池架构**，动态开辟内存块，将碎片集中处理，

增加程序空间局部性。
这样一来，对于现代 CPU，提高了各级缓存的命中率，也使得 CPU 的各种现代特性——预取(pre-fetch)、指令级并行(Instruction Level Parallelism)等得以被充分利用。

2）面对大文件 html 绘制工作量大，程序卡顿
解决方法：使用“惰性”方式求值：如果没有改变标签页、没有用户输入，就不必从头解析生成 Render Tree，而是可以直接利用。
滚动时的溢出可以通过先绘制渲染结果，再通过 UI 遮罩覆盖的方式解决。

3）Windows 默认栈空间相对（部分）Linux 下较小，面对大文件建树时递归过深容易栈溢出发生段错误。
解决方法：通过调整编译选项，指定、扩大预留的栈空间。

4）小段代码大量调用引起性能损失，代码复用性差
解决方法：灵活使用 inline 与宏，利用宏的字符串拼接实现类似“魔术方法”的功能，在 C 语言中进行此类封装尤其有利于提高鲁棒性与可复用性。例如,通过定义 SAVEPEN 与 RECOVERPEN 的宏快速保存画笔状态(不含位置)，极大减少了代码量；对于简单函数利用 inline 特性直接展开，免去了入栈出栈开销。

B

1） 编写 UI 时，对库里已有的函数不够熟悉，无法为满足个性化功能做出更改
解决方法：回到库中的.c 文件里仔细研读每个函数的具体实现方式，必要时修改库里的函数。

2） 设计颜色时，不了解颜色搭配的原则，不清楚网页布局的设计方法
解决方法：上网查找色彩搭配和网页设计的相关理论和实例，加以改编应用。最终选用两套莫兰迪配色方案，分别为橙色系暖色、蓝色系冷色。

C

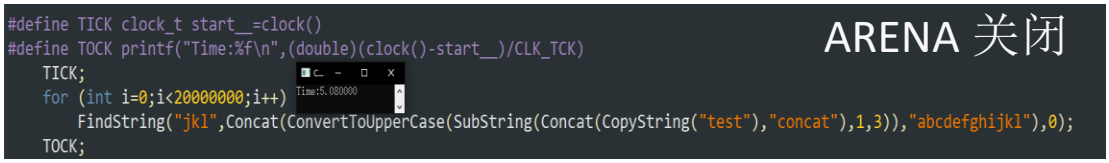
1） 挑战：网页的前进后退
解决方案：双向链表，通过 pre 和 next 方法实现前进和后退。

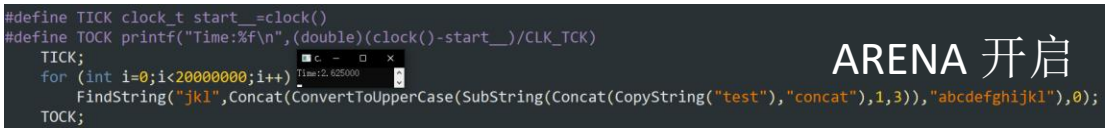
2） 挑战：不同位置标签页的新建，关闭，以及前进后退
解决方案：建立多个双向链表，通过链表节点内容的复制，不改变总共的链表个数，及时释放无用节点。

5.7 合作总结

5.7.1 开发亮点

- 前后端分离，通过细分功能，贯彻模块化、层次化的思想，**自顶向下设计，自下而上实现；**
- 通过**内存池**统一调度内存，增加代码的空间局部性，激活现代 CPU 特性加速访存速度；**访存加速比 2x,总加速比达 1.3x.**





- 配色精美，UI 统一和谐，美观程度高，
- **前后端分离**，模块低耦合，便于对接；
- 承上启下，合作过程积极交流讨论，结合后续 CS 课程内容（如编译原理、模块化与面向对象思想、计算机系统、自顶向下计算机网络），在实践中感受计算机各类思想、模式(pattern)之美；
- 贴合实际浏览器渲染 HTML 流程，**本次实践中除将 CSSDOM 树简化为线性结构外严格遵循标准流程**：构建 DOM 树后结合 CSS 生成渲染树，遇到特定事件时执行**回流重绘**，程序**可拓展性强**，添加新功能时对接方便；
- **树状结构**使得元素定位、修改方便，也由此实现了 **CSS 对父级元素属性的继承与标签的嵌套效果；**
- 能够解析 CSS 的一个子集，识别合法的**标准 CSS 语法**并应用效果。

5.7.2 开发历程

2021 年 4 月 21 日：小组成员初次见面，分析评估各选题的技术难点和可取度

2021 年 4 月 28 日：确定选题为简易浏览器。

2021 年 4 月 28 日——2021 年 5 月 17 日：独立完成各自的任务。期间通过线上进行交流与互相答疑，提升自我。

2021 年 5 月 17 日：线下验收前期工作成果，交流难点，商讨后期任务分工。

2021 年 5 月 24 日：组长于凌晨大战 Token 之间复杂的状态转移。

2021 年 5 月 25 日：组长再于凌晨大战各种神秘 bug。

2021 年 5 月 26 日:三位成员在组长工位于晚上 10 点奋战到次日凌晨 5 点，UI 基本成型，HTML 解析器框架也基本完工。

2021 年 5 月 27 日：组长挑灯夜战，不断完善，进行最终的整合冲刺。

2021 年 5 月 28 日：万事大吉，微调后交付。

6 收获感言

A（组长）

本课程的大作业是我自大学以来接触的第一个团队分工合作项目，我也付出了超乎自己想象的热情。我不希望这次作业沦为在画图上钻牛角尖的原地踏步，而是希望这次作业能够继往开来，既是一次“复健”，也是初窥编译原理、计算机系统架构或是软件工程的一块敲门砖。

“痛并快乐着”也许是对这次作业最好的总结。但是在痛苦以外，第一次看见 UI 成型，第一次看见自己写的 HTML 解析器打出五彩斑斓，以各种字体、姿态翻飞的"hello world"时，那样的感动是无以言表的。我想对于许多人来说，是时候暂时抛开种种过于华丽的编程概念了：不如静下心来打磨最基本的编程技能，享受蕴藏在 C 语言设计中、经过历史长河积淀下来的编程智慧——以设计模式的形式。享受最纯粹、根源、过程式的对编码的乐趣，保持对计算机朴素的热爱才是最重要的。

此外，我深深地感激陪我奋战的两位队员。他们勤奋聪慧，我觉得有这样的合作机会实在是一种荣幸：比如聚众写代码熬穿肯定是大一学年最难忘的事情之一了。

B

之前我一直以为我所学的学科就是枯燥的编写一串串代码，不断失望的 debug 的过程。但是这次完成大程作业，我主要负责 UI 的编写，看着显示界面不断地变得丰富美观，能够通过增设的按钮实现越来越多的功能，心中的成就感是难以言表的。在设计页面颜色搭配的过程中，我去找了相关的色彩学的文献和书籍，不断微调 rgb 参数，来搭配出更和谐的颜色，制造出页面颜色渐变的效果，最终实现了“sea”系，和“sun”系的两套页面风格，并增加按钮实现了自由切换。

很幸运此次大作业能和其他两位小组成员组成一队，他们的友善、耐心、风趣、谦虚，打破了我对这个行业的刻板印象。在专业能力上，我远不及二位大佬，但他们一直包容、帮助和鼓励着我，让我非常感激。我从他们身上学到了很多，更加了解了计算机这门学科，也对它由惧怕变得多了几分热爱。

这次的团队合作我由于自 身能力所限，没有涉及到核心功能的实现，但编写 UI 的过程让我对库里的函数更加熟悉，更加具有攻坚克难的精神和细心耐心的品质，也在团队中学会承担责任和分工合作。

C

通过这次大作业的团队合作，我发现眼界决定世界这句话真的很有道理，在被组长大佬带飞的过程中，见识到许多从未听过的名词，许多从未接触的语法和计算机知识，这种震撼感和差距感深深地影响着我。我和 B 同学一样，专业知识方面可能有所欠缺，所以在实现了网页路由和标签页之后就去协助她一起搞 UI 了，两位同伴都给我带来了很美好的回忆，组长的强硬实力和刻苦，B 同学的细

心和耐心，让我在合作过程中重新审视自己，重新认识自己，知道自己的不足。当一个人的眼界突然开阔，真正接触到那些比你优秀的人，你才会知道自己和理想的差距有多大，许多人总是在舒适圈停滞不前的时间里愈渐陷入彷徨，只有打破这层桎梏，见到更优秀的人他们在做什么，他们的世界是怎么样的，才能有走出舒适圈，追求更好更优秀的自己的动力和勇气。

7 参考资料

- 芝芝心理.色彩心理学：生活中，红、粉、橙、黄、绿暗藏的心理和使用方法[OL].(https://baijiahao.baidu.com/s?id=1679375687678313462). 2020-10-02
- 编译原理 [EB]. Alfred V.Aho 等.北京：机械工业出版社,2009 第二版.68 页
- Ilya Grigorik. Render-tree Construction, Layout, and Paint [OL].(https://web.dev/critical-rendering-path-render-tree-construction/).2018-08-17
- zhaohai-shen. LevelDB Arena [OL].(https://www.jianshu.com/p/aec6f7aacf5a). 2014-06-05
- @籴粜@.SGI STL 源码剖析——空间配置器[OL].(https://blog.csdn.net/qq_31082639/article/details/122650384). 2022-01-29
- 龚奕利,贺莲[译].深入理解计算机系统 [EB].Randal E. Bryant 等[著]. 北京：机械工业出版社,2016.7 第三版.第 357 页.
- kaka. 一起来写个简单的解释器（四）：词法分析器 Lexer [OL].(https://zhuanlan.zhihu.com/p/378612047). 2021-06-24.